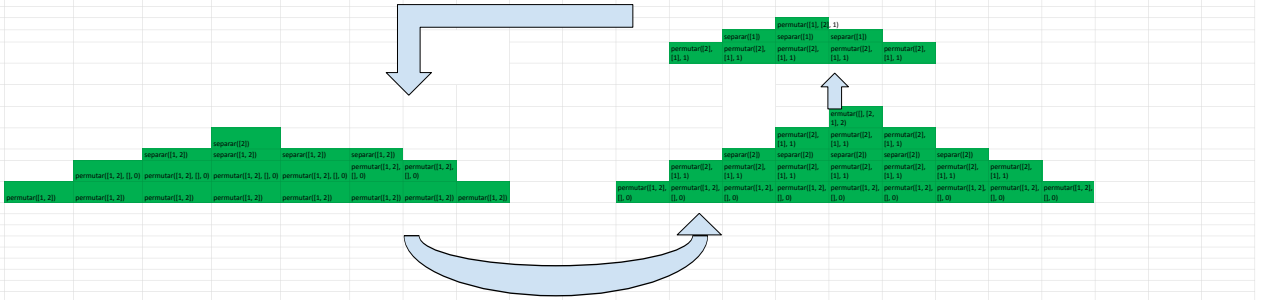
EJERCICIO 1			
N	CONDICION (n == 0)	LLAMADA RECURSIVA	RETURN
4	F	factorial (3)	-
3	F	factorial (2)	-
2	F	factorial (1)	-
1	F	factorial (0)	-
0	V	-	1
1	-	-	1x1= 1
2	-	-	2x1= 2
3	-	-	2x3= 6
4	-	-	4x6= 24

EJERCICION 2			
LISTA	CONDICION (VACIA)	LLAMADA RECURSIVA	RETURN
[a,b,c,d]	f	contar([b,c,d])	-
[b,c,d]	f	contar([c,d])	-
[c,d]	f	contar([d])	-
[d]	f	contar([])	-
[]	v	-	0
[d]	-	-	1+0=1
[c,d]	-	-	1+1=2
[b,c,d]	-	-	1+2=3
[a,b,c,d]	-	-	1+3=4

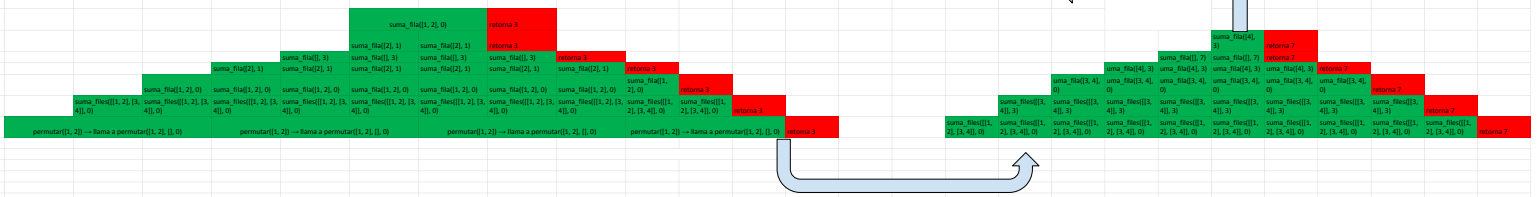
Lista (entrada)	n	acum	Acción / Salida
[:a, :b]	0	[]	Nivel 0 -> tomando :a
[:b]	1	[:a]	Nivel 1 -> tomando :b
[]	2	[:b, :a]	Caso base → imprime [:a, :b]
[:a]	1	[:b]	Nivel 1 -> tomando :a (segunda rama del for)
[]	2	[:a, :b]	Caso base → imprime [:b, :a]

```elixir
# Caso base
defp permutar([], acum, n) do
  IO.puts("Nivel #{n} -> #{inspect Enum.reverse(acum)}")
  [Enum.reverse(acum)]
end

# Caso recursivo
defp permutar([h | t], acum, n) do
  IO.puts("Nivel #{n} -> tomando #{h}")
  for {elem, rest} <- separar([h | t]) do
    permutar(rest, [elem | acum], n + 1)
  end
  |> List.flatten()
end
```



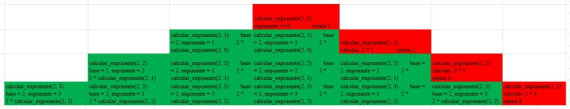
Entrada (fila/resto)	acc	Acción / Salida
[[1,2],[3,4]]	0	Llama suma_filas con fila=[1,2]
[1,2]	0	Llama suma_fila con h=1
[2]	1	Llama suma_fila con h=2
[]	3	Caso base → retorna 3
[[3,4]]	3	Llama suma_filas con fila=[3,4]
[3,4]	0	Llama suma_fila con h=3
[4]	3	Llama suma_fila con h=4
[]	7	Caso base → retorna 7
[]	10	Caso base final → Salida total: 10

```elixir
# Caso base
defp suma_filas([], acc), do: acc


defp suma_filas([fila | resto], acc) do
  suma_filas(resto, acc + suma_fila(fila, 0))
end

# Suma recursiva
defp suma_fila([], acc), do: acc
defp suma_fila([h | t], acc), do: suma_fila(t, acc + h)
end
```



Base	Exponencial	Exponencial == 0	calcular_exponente
2	3	False	2 * calcular_exponente(2, 3-1)
2	2	False	2 * 2 * calcular_exponente(2, 2-1)
2	1	False	2 * 2 * 2 * calcular_exponente(2, 1-1)
2	0	True	2 * 2 * 2 * 1 = 8

```elixir
def calcular_exponente(_base, 0), do: 1

def calcular_exponente(base, exponente) do
  base * calcular_exponente(base, exponente - 1)
end
```



elemento	lista	head	coincidencia?	do
3	[1, 2, 3, 4]	1	False	buscar_elemento(elem, tail)
3	[2, 3, 4]	2	False	buscar_elemento(elem, tail)
3	[3, 4]	3	True	True

```
def buscar_elemento(_elem, []), do: false

def buscar_elemento(elem, [elem | _resto]), do: true

def buscar_elemento(elem, [_otro | resto]) do
  buscar(elem, resto)
end
```

		buscar_elemento(3, [3, 4]) elem = 3, head = 3 return true
	buscar_elemento(3, [2, 3, 4]) elem = 3, head = 2, tail = [3, 4] buscar_elemento(3, [3, 4])	buscar_elemento(3, [2, 3, 4]) elem = 3, head = 2, tail = [3, 4] buscar_elemento(3, [3, 4])
buscar_elemento(3, [1, 2, 3, 4]) elem = 3, head = 1, tail = [2, 3, 4] buscar_elemento(3, [2, 3, 4])	buscar_elemento(3, [1, 2, 3, 4]) elem = 3, head = 1, tail = [2, 3, 4] buscar_elemento(3, [2, 3, 4])	buscar_elemento(3, [1, 2, 3, 4]) elem = 3, head = 1, tail = [2, 3, 4] buscar_elemento(3, [2, 3, 4])